(12) UK Patent Application (19) GB (11) 2 296 799 (13) A

(43) Date of A Publication 10.07.1996

(21) Application No 9500252.3

(22) Date of Filing 06.01.1995

(71) Applicant(s)
International Business Machines Corporation

(Incorporated in USA - New York)

Armonk, New York 10504, United States of America

(72) Inventor(s)
Michael Platt
Andrew James Stanford-Clark
Graham Derek Wallis

(74) Agent and/or Address for Service
D R Horner
IBM UK Ltd, Mailpoint 110, Hursley Park,
WINCHESTER, Hampshire, SO21 2JN,
United Kingdom

(51) INT CL^6
G06F 17/30

(52) UK CL (Edition O)
G4A AUDB

(56) Documents Cited
EP 0625756 A1

(58) Field of Search
UK CL (Edition N) G4A AUDB
INT CL^6 G06F

(54) Processing parallel data queries

(57) A system for processing requests from a client computer (300) for retrieval of data from a database (320) having a plurality of database servers (330, 450, 340, 350) on which data is stored, comprises: a storage means (430) for storing information identifying how the data is distributed across the plurality of database servers; a first processing means (440), interposed between the client computer (300) and the database (320), for receiving a request from the client computer in a standard query language, and with reference to the storage means (430), transforming the request into one or more constituent requests in the standard query language; and communication means (445; 415) for transmitting the constituent requests to the associated database servers. Additionally a second processing means (440; 410, 420) is arranged to receive the data sent from the database servers as a result of the constituent requests, and to assimilate that data as a single response for communication back to the client computer. This provide a high performance query server offering transparency to the client, heterogeneous operation, isolation between applications and data, and a choice of parallel architectures.

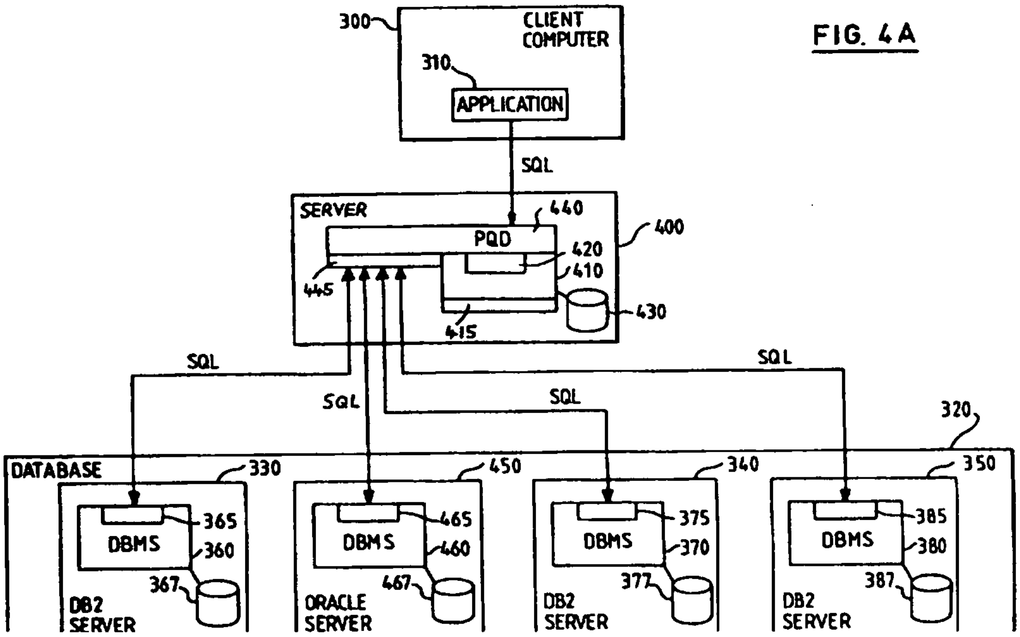


FIG. 4A

GB 2 296 799

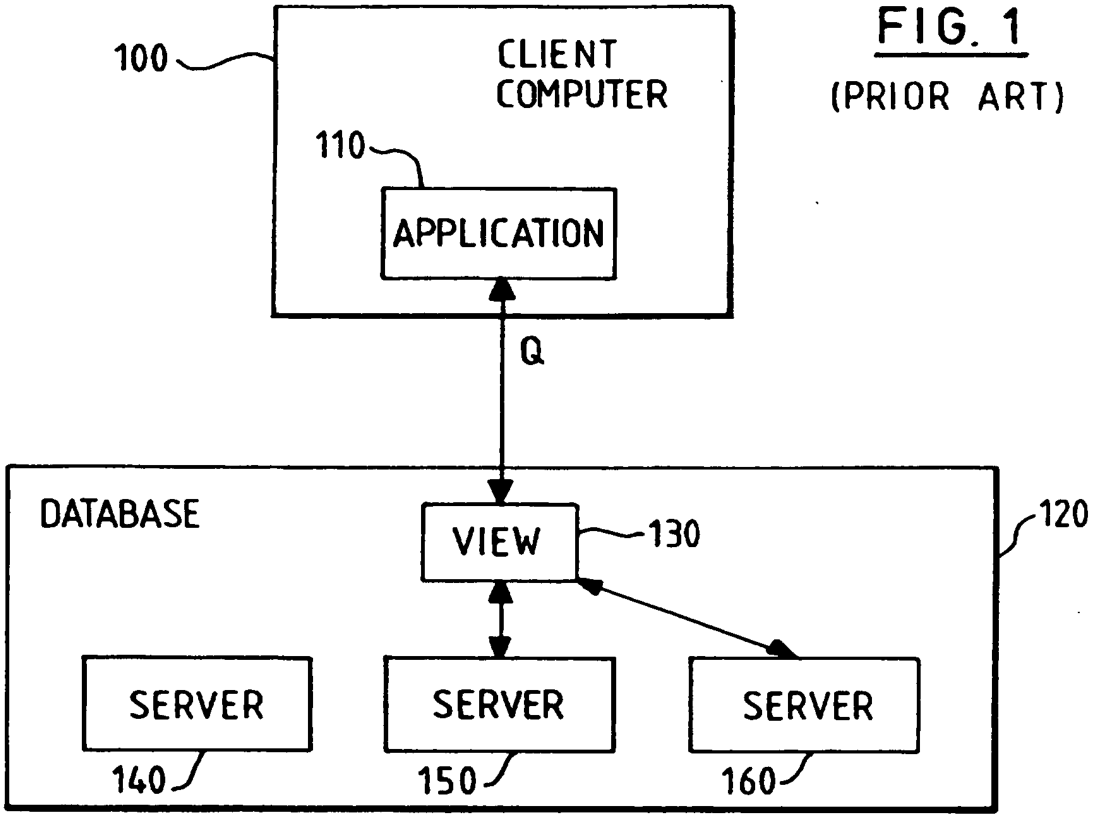


FIG. 1
(PRIOR ART)

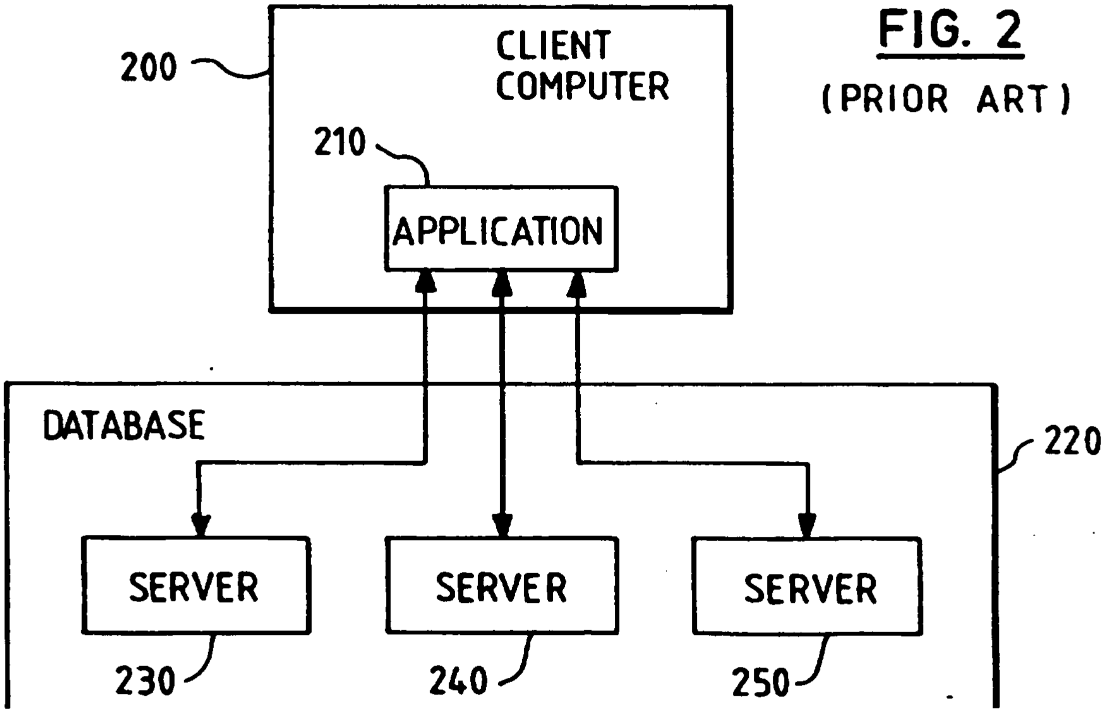


FIG. 2
(PRIOR ART)

FIG. 3

(PRIOR ART)

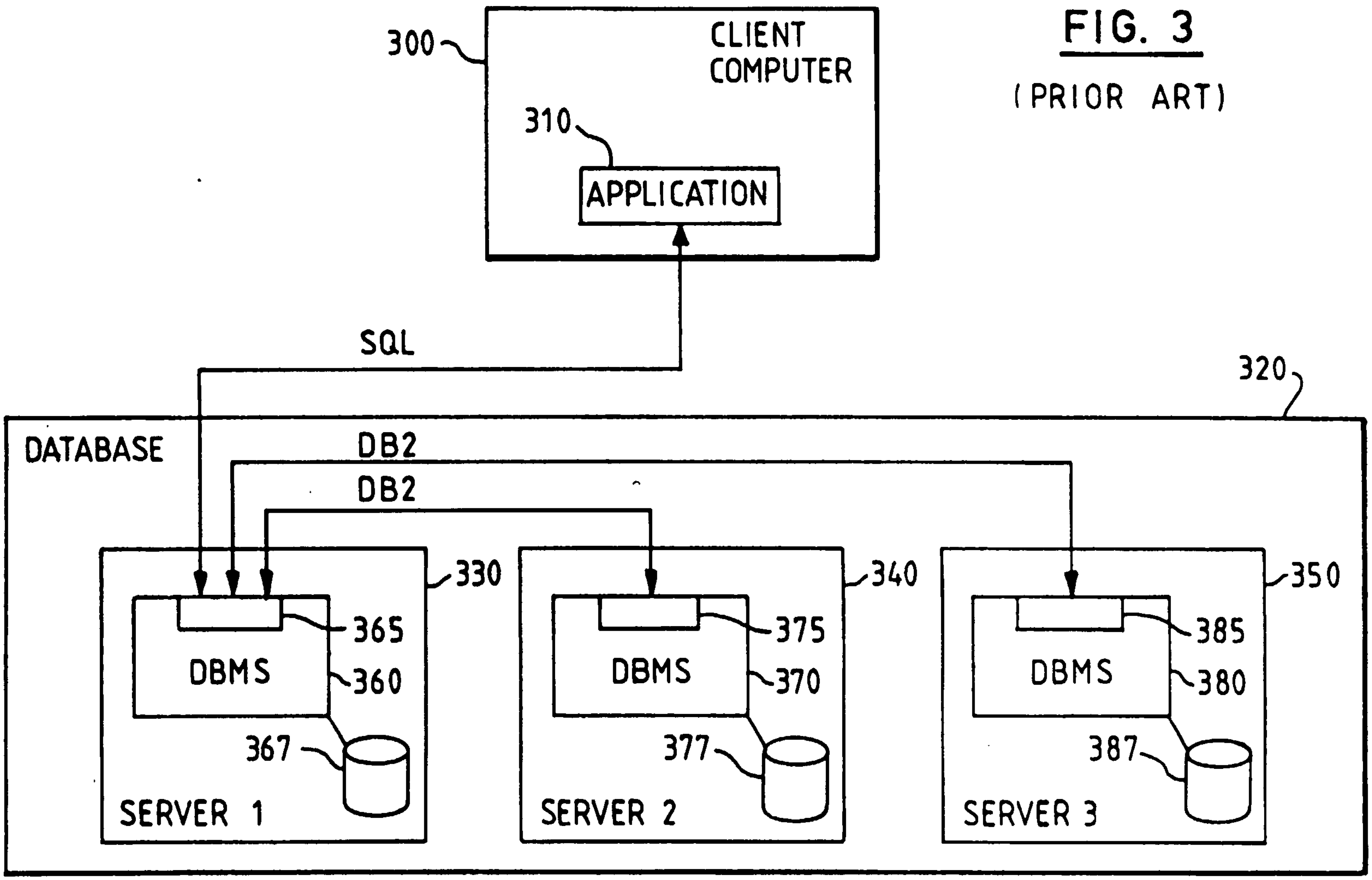


FIG. 4A

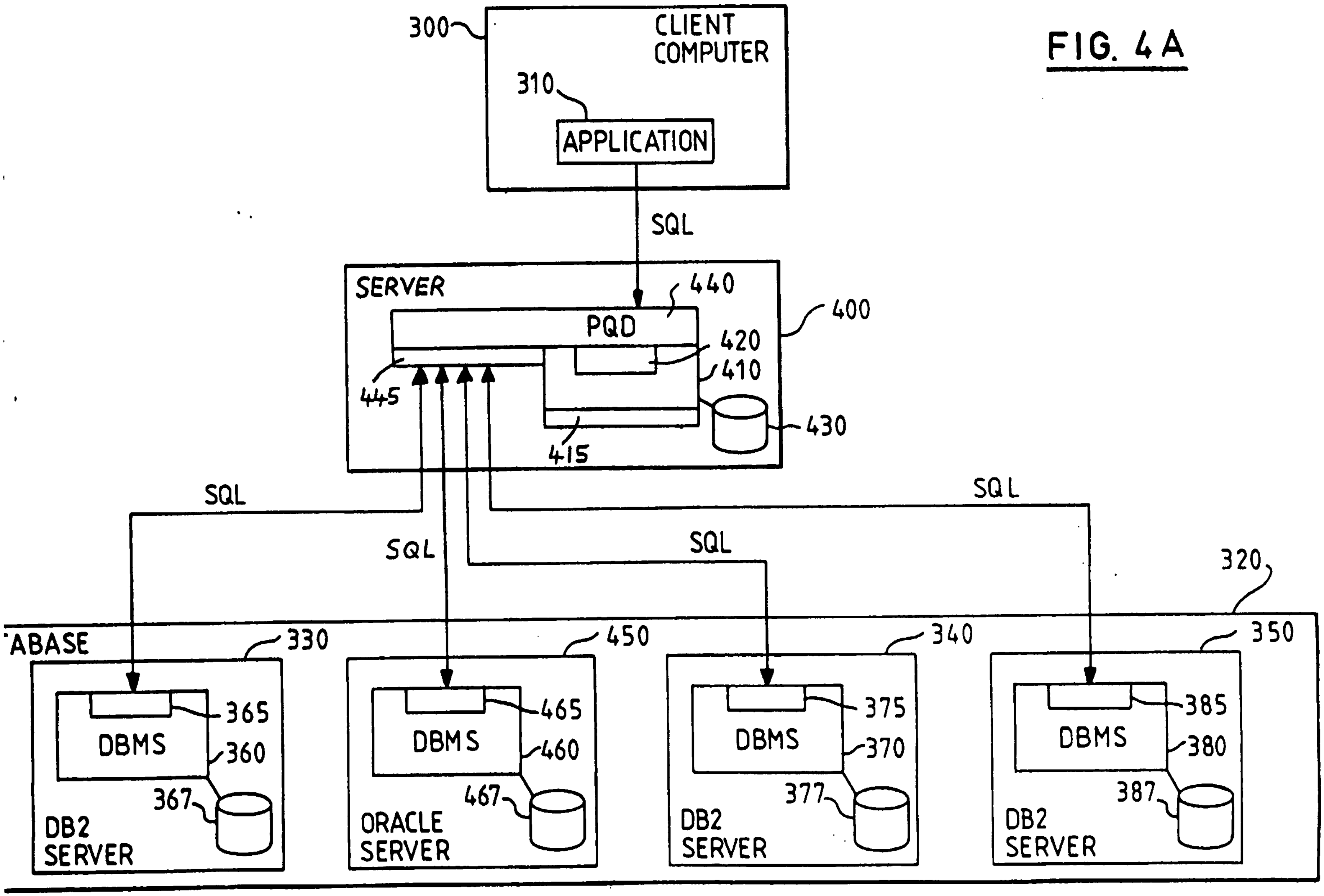
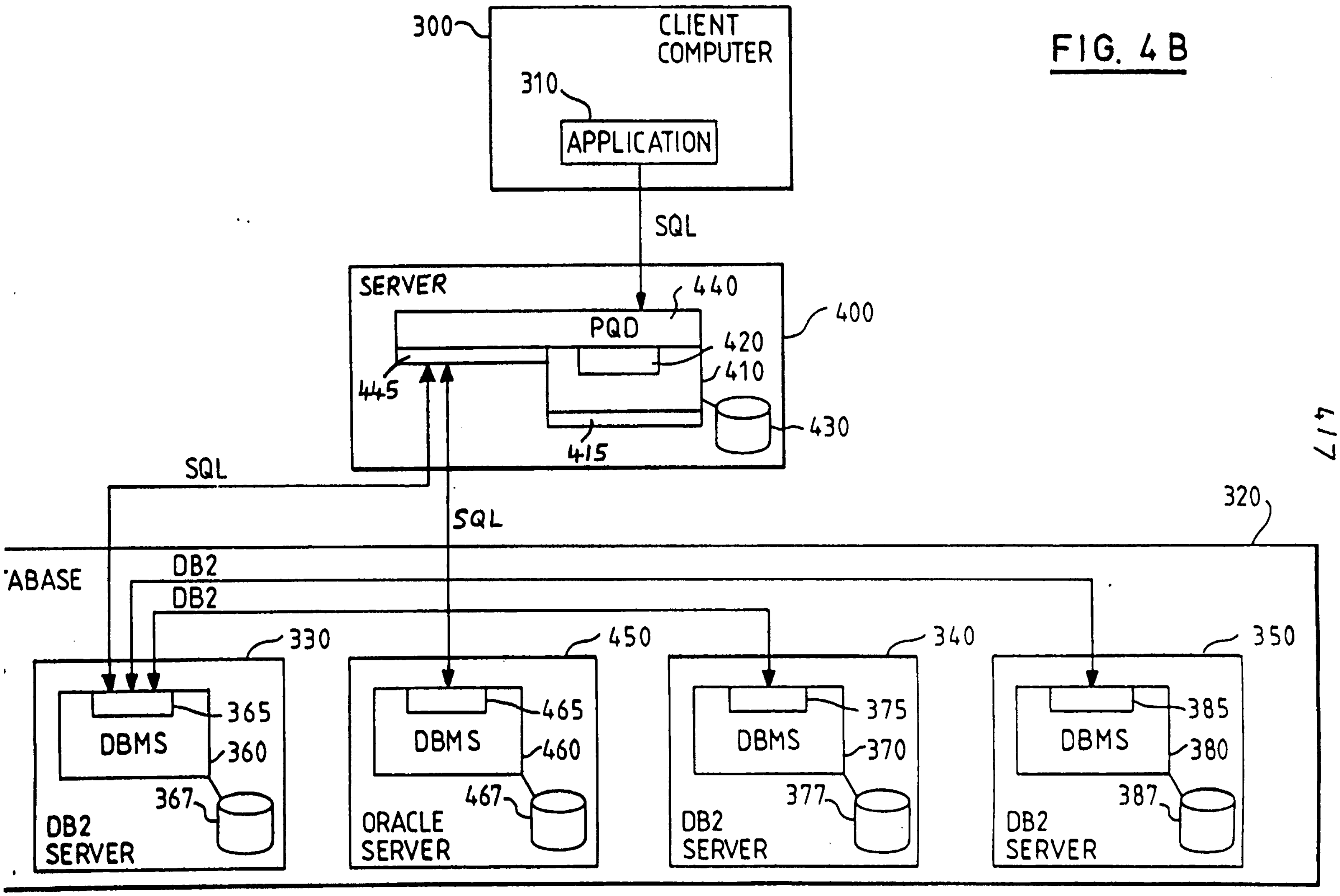

300
CLIENT COMPUTER
310
APPLICATION
SQL
SERVER
440
PQD
400
420
410
445
430
415
SQL
SQL
SQL
SQL
320
ABASE
330
450
340
350
365
DBMS
360
367
DB2 SERVER
465
DBMS
460
467
ORACLE SERVER
375
DBMS
370
377
DB2 SERVER
385
DBMS
380
387
DB2 SERVER

FIG. 4B


300
CLIENT COMPUTER
310
APPLICATION
SQL
SERVER
440
PQD
400
420
410
445
430
415
SQL
SQL
320
ABASE
DB2
DB2
330
450
340
350
DBMS
365
360
367
DB2 SERVER
DBMS
465
460
467
ORACLE SERVER
DBMS
375
370
377
DB2 SERVER
DBMS
385
380
387
DB2 SERVER

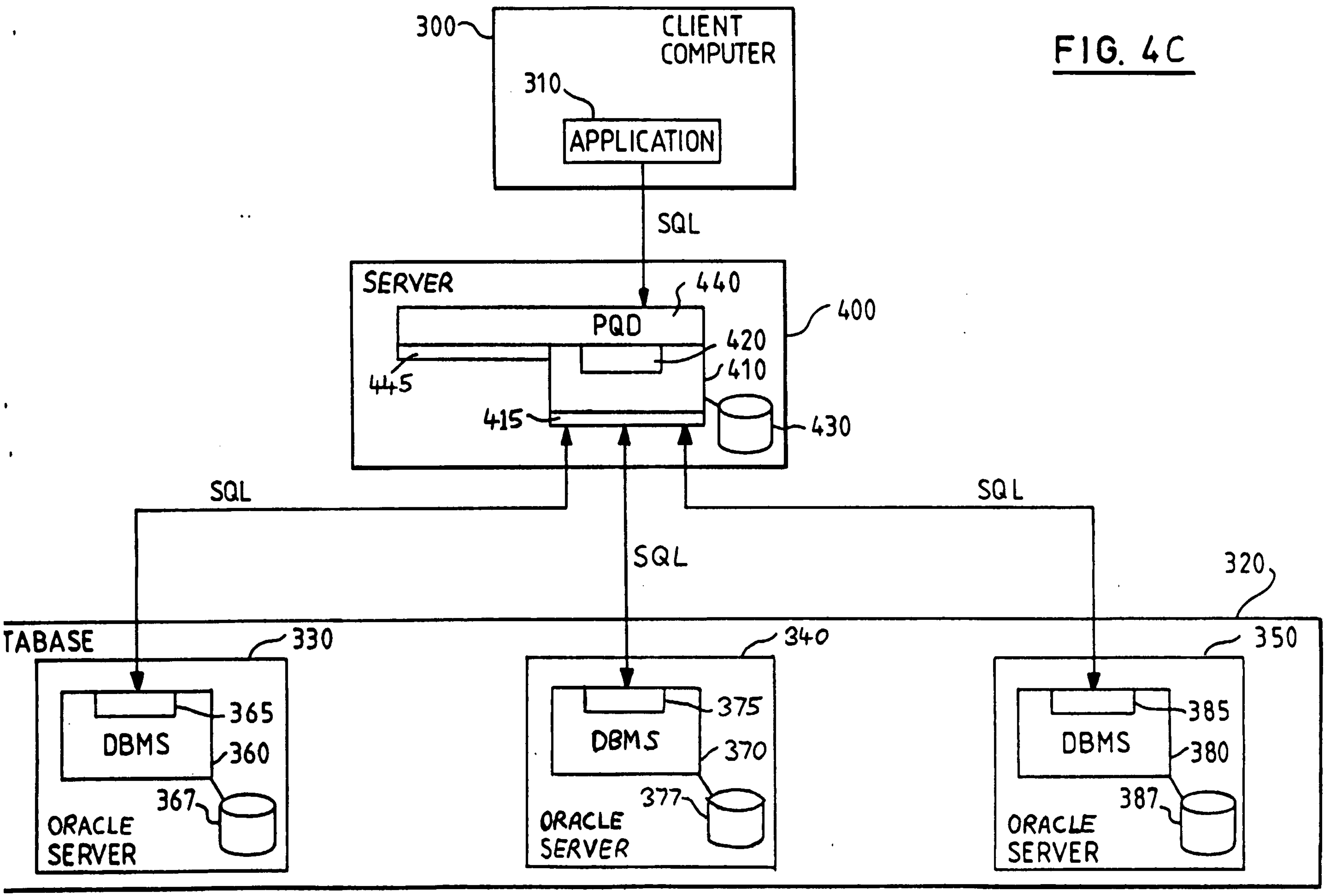

300
CLIENT COMPUTER
310
APPLICATION
SQL
SERVER
440
PQD
400
420
410
445
415
430
SQL
SQL
SQL
320
TABASE
330
340
350
DBMS
365
360
367
ORACLE SERVER
DBMS
375
370
377
ORACLE SERVER
DBMS
385
380
387
ORACLE SERVER

FIG. 4C

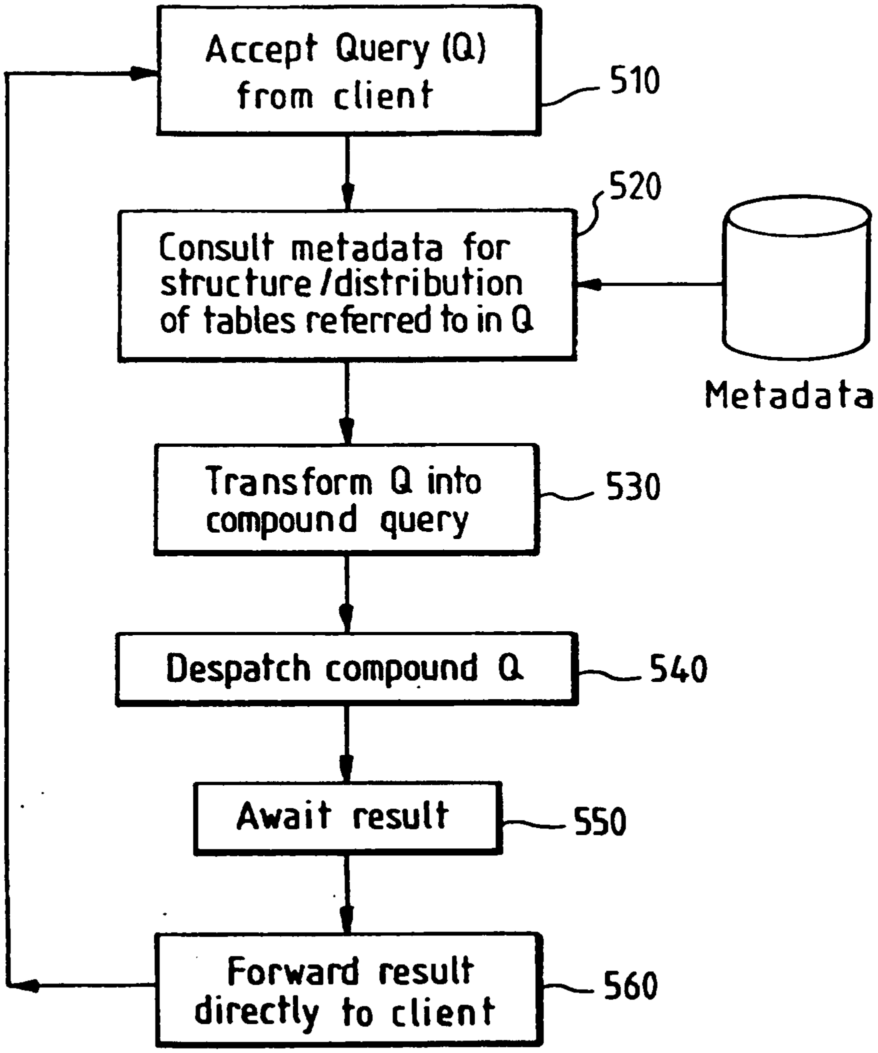

Accept Query (Q) from client
510
520
Consult metadata for structure /distribution of tables referred to in Q
Metadata
Transform Q into compound query
530
Despatch compound Q
540
Await result
550
Forward result directly to client
560

FIG. 5A

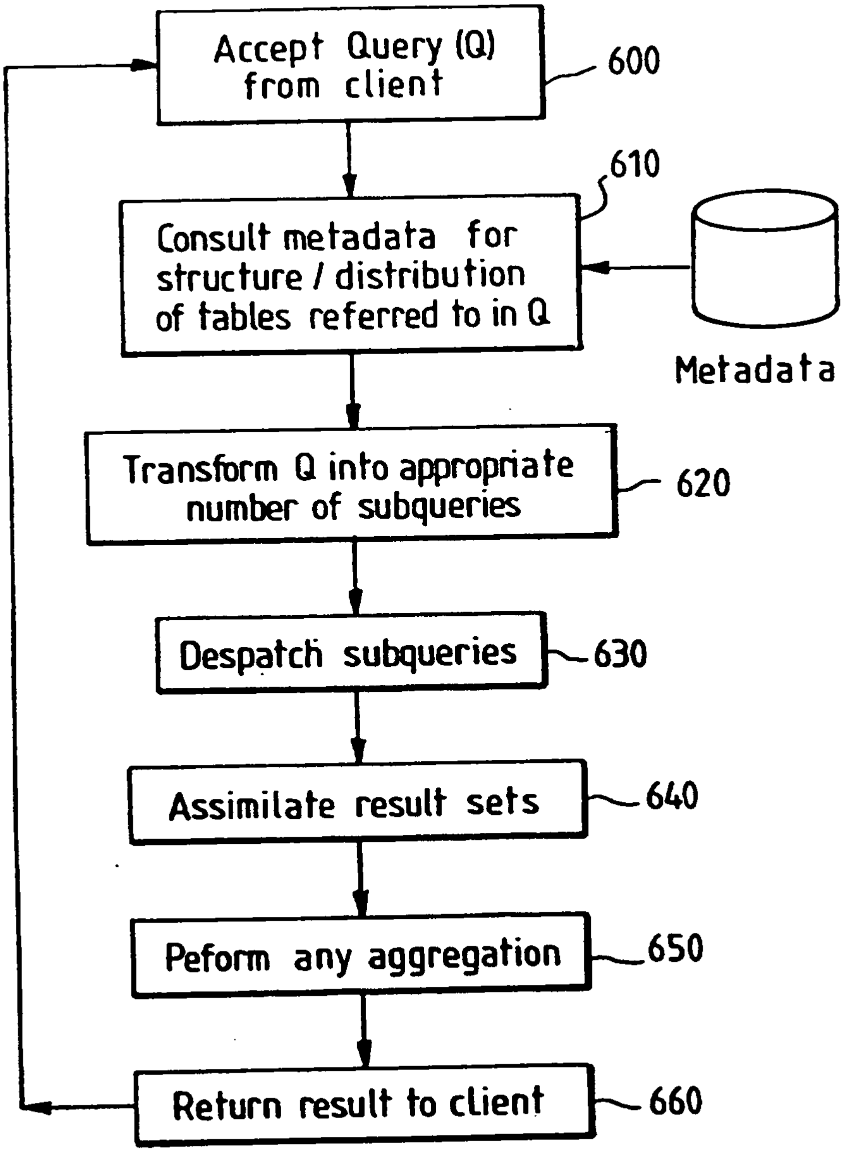

Accept Query (Q) from client
600
Consult metadata for structure / distribution of tables referred to in Q
610
Metadata
Transform Q into appropriate number of subqueries
620
Despatch subqueries
630
Assimilate result sets
640
Peform any aggregation
650
Return result to client
660

FIG. 5B

A SYSTEM AND METHOD FOR PROCESSING PARALLEL DATA QUERIES

The present invention relates to the area of parallel processing. More specifically, the present invention relates to a system and method for processing parallel data queries, and in particular for processing requests from applications on a client computer for retrieval of data from a database having a plurality of database servers.

Parallel processing is the harnessing of multiple processors to work in unison on a problem. This can provide significant performance benefits over the use of a single processor in cases where the computing power required to solve the problem is relatively large compared to the computing power of a single processor.

One such case is that of decision support, decision Support being the term applied to the use of processor-intensive queries of large databases in order to analyze historic data and support the development of future strategy. This is also sometimes referred to as MIS (Management Information Systems), EIS (Executive Information Systems) or Data Mining. Corporate databases are now becoming so large, and decision support queries so complex, that the queries take an unacceptable length of time to execute, in some cases taking several days to complete.

Decision Support queries on large databases are a good example of an area in which parallel processing can be used to improve performance, and there are a number of techniques which have been used to try and accelerate the processing of complex queries, based broadly on the concept of using parallel processing. However, they all suffer from various weaknesses, as will become apparent from the following discussion. In the following discussion, the term "parallel database" is used to refer to both a closely coupled machine, or group of machines, in which secondary (eg. disk) and/or primary (e.g. memory) storage mechanisms are shared (commonly referred to as a single parallel database), and also to a group of machines, or processors within a single machine, which are loosely coupled and do not share either primary or secondary storage mechanisms (commonly referred to as a distributed database).

One parallel processing technique known in the prior art involves parallel database. Whole tables may be allocated to a single server where the tables are small enough to each fit within one server, or tables may each be partitioned across a number of servers in cases where there is too much data in a table for it to be accommodated by a single server. The latter occurs frequently where large corporate databases are being migrated from large (mainframe) servers to clusters of mid-range servers. Following such a partitioning of tables, any queries made against the data can be allocated across the available nodes of the parallel database.

One way in which this can be achieved is by the use of "views". A "view" is a definition stored in the database by the database administrator (or "DBA") which is then executed by querying the view directly. Such a query produces a result set which is dynamic in the sense that it is only created at execution time. Figure 1 illustrates how the 'view' technique is used to process queries from a client computer to a database. When an application 110 is run on a client computer 100, it periodically sends a data query Q to a server of the database 120. To enable this query to be processed, the user of the application 110 will need to give details of the query Q to the DBA to enable the DBA to construct a view definition 130. Once this has been done the query Q is always sent directly to the server containing the view 130. This is then executed to extract the necessary data from the various database servers 140, 150, 160 within the database, and to send the resultant data back to the application 110. If the DBA alters the structure of data within the database, then he/she will update the view definition accordingly. The application itself need know nothing about the database structure since it always queries the view 130.

With regard to the tables of data stored statically in the database 120 it is usual practice to establish an index for the various tables so as to improve the speed of access to data in the tables. This benefit is particularly marked when the table is large, since without the index the table entries need to be accessed sequentially until the desired entry is reached. However, since the table of data created by the "view" is dynamic it is not possible to establish an index for the view. With regard to queries against data allocated across a number of the database servers, the views created can be very large, and the inability to use indices is a major disadvantage of the viewing approach, rendering such a technique of limited applicability.

As an alternative to the "view" approach, allocation of partitioned queries could be achieved by re-writing an application program such that it submitted multiple queries to the various database servers, and combined the results (e.g. via the UNION operator). This type of approach is shown schematically in Figure 2, in which an application 210 within a client computer 200 is written such that it sends individual queries to the database servers 230, 240, 250 within the database 220.

This arrangement has the undesirable consequence that the application writer must know the structure of the data and the manner in which it has been partitioned across the servers 230, 240, 250. Further, it is not a practical long-term solution because it ties the application code to the structure of the database, and hence fixes that structure. The data structure within the database may need to be changed, for example when a table grows to a size which can no longer be accommodated by a single database server, and recoding the application each time such changes are necessary would be prohibitively expensive.

A more sustainable approach to solving the problem is to use some form of utility (commonly referred to as an 'optimiser') which sits on top of what is traditionally referred to as the Database Management System (DBMS) of a database server, and which decomposes a query, despatches the resultant subtasks and coordinates reassembly of the results of those subtasks. This approach will be discussed further with reference to Figure 3, which illustrates how an optimiser is used to handle parallel queries in a homogeneous database, ie a database in which all of the servers are of the same type (eg all IBM DB2, all Oracle 7, etc).

Figure 3 shows a database 320 consisting of three IBM DB2 servers 330, 340, 350. In this particular instance the parallel database is a distributed database since the secondary storage devices 367, 377, 387 are not shared by the servers. However the discussion is equally applicable to the situation in which a single parallel database is used instead of a distributed database. Each server in the database has a DBMS 360, 370, 380, and each DBMS includes an optimiser 365, 375, 385. An application 310 being run in the client computer 300 is programmed to send its data queries in a standard query language such as SQL to Server 1 (server 330), where they are received by the optimiser 365.

The storage device 367 forming part of Server 1 not only has tables of data stored, but also includes a data dictionary containing information about the data contained in the other servers 340, 350 of the database 320 (this information being referred to hereafter as 'metadata'). As an example, if the three servers 330, 340, 350 are used to store a list of all the employees of a company along with their personnel details, the metadata in the data dictionary of storage device 367 may indicate that those employees with surnames A-H are stored on server 1, those employees with surnames I-P are stored on server 2, and those employees with surnames Q-Z are stored on server 3.

The optimiser uses such metadata to decompose the SQL query into a number of subtasks expressed in DB2 format, which are then distributed to the other servers for execution. Once all of the servers have performed the tasks directed to them, the results are returned to the optimiser 365, where they are reassembled into one query result for dispatch to the application 310. The manner in which the query result is packaged for return to the client application 310 will typically depend on the DBMS being used.

This approach has been taken by a number of database vendors, some of whom have made use of features of particular database systems (e.g. internal striping of tables), and have used the DBMS optimiser's ability to access the data dictionary of the database server to control the decomposition of complex queries. These techniques suffer from the drawback that they are limited to the DBMS system on which they have been implemented, accepting a complex query in a standardised query language such as SQL, and converting it (decomposing it) into a number of queries which are only defined within the internal representation of the DBMS used (eg IBM DB2). They cannot therefore operate on cross-system queries where more than one type of database server is used within a parallel database. The object of the present invention is to alleviate this problem.

Accordingly the present invention provides a system for processing requests from a client application on a client computer for retrieval of data from a database having a plurality of database servers on which data is stored, the system comprising: a storage means for storing information identifying how the data is distributed across the plurality of database servers; a first processing means, interposed between the client computer standard query language, and with reference to the storage means, transforming the request into one or more constituent requests in the standard query language; communication means for transmitting the constituent requests to the associated database servers; a second processing means being arranged to receive the data sent from the database servers as a result of the constituent requests, and for assimilating that data as a single response for communication back to the client computer; and dispatch means for sending the single response to the client application.

The present invention uses the power of parallel processing to create a high performance query server which also offers transparency to the client, heterogeneous operation, isolation between applications and data, and a choice of parallel architectures. The combination of these attributes are not possible using any of the prior art techniques.

The query decomposer of the present invention accepts standard sequential queries, and automatically detects any data parallelism present in the structure of the database. Where such parallelism exists the decomposer transforms the original query such that the underlying database query optimiser can take advantage of the parallelism inherent in the data.

The processing means used in the system of the present invention is not part of any database server DBMS. It is an example of what is loosely referred to in the industry as "middleware". That is, it fits in between the application (top) and database server DBMS (bottom). Anything that fits into the gap between the two is referred to as middleware, with notable examples being load balancing, transaction monitoring, security services, etc.

In preferred embodiments, the first processing means produces one constituent request for each type of database server in the database, rather than producing one constituent request for every server in the database that has data of the type requested. For example, if the database consists of three IBM DB2 database servers and one Oracle 7 database server, then the first processing means will produce two constituent requests. With such an arrangement, the associated database server then further divides the constituent request so as to send one request to each database server of the same type as the associated server to send such requests only to database servers of the same type which have data of the type requested. In preferred embodiments, the associated database server further recombines results from the database servers of the same type before sending the data to the second processing means.

If all the database servers in the database are of the same type, then the first processing means preferably uses aliases stored in the storage device to create the constituent requests. In the Oracle 7 environment such aliases are called 'synonyms', but equivalent representations typically exist in other database systems. If, on the other hand, some of the database servers in the database are of different types, then the first processing means uses metadata (of which aliases may form a part) stored in the storage device to create the constituent requests. The metadata will typically include the names of any tables which have been partitioned, along with the location of those partitions within the servers of the parallel database.

Preferably the second processing means also comprises aggregation means for performing any necessary aggregation of the data received by the second processing means from the database servers.

Viewed from a second aspect the present invention provides a method of processing requests from a client application on a client computer for retrieval of data from a database having a plurality of database servers on which data is stored, the method comprising the steps of: storing information in a storage means for identifying how the data is distributed across the plurality of database servers; employing a first processing means, interposed between the client computer and the database, to receive a request from the client computer in a standard query language, and with reference to the storage means, transforming the request into one or more constituent requests in the standard query language; transmitting the constituent requests to the associated database servers; receiving the data sent from the database servers as a result of the constituent requests, and assimilating that data as a single response for communication back to the client computer; and sending the single response to the client application.

The present invention will be described further, by way of example only, with reference to preferred embodiments thereof as illustrated in

Figure 1 illustrates how 'views' are used in prior art database systems to process queries from client applications;

Figure 2 illustrates another prior art technique in which client applications are re-written to reflect the structure of a database;

Figure 3 illustrates a prior art technique in which an optimiser within the DBMS of a database server is used to decompose queries when the database servers within the database are all of the same type;

Figures 4A, 4B and 4C are block diagrams illustrating how three embodiments of the present invention process queries from client applications;

Figure 5A is a flow diagram illustrating the process carried out by one embodiment of the present invention where the database is homogeneous; and

Figure 5B is a flow diagram illustrating the process carried out by a second embodiment of the present invention where the database is heterogeneous.

The arrangement of the preferred embodiment of the present invention will now be discussed with reference to Figure 4A, which is a block diagram showing the elements of the preferred embodiment. In Figure 4A a client computer 300, which may for example be a Personal Computer, is connected to a server 400. The server 400 is in turn connected to a database 320 consisting of a number of parallel database servers 330, 450, 340 and 350. Each of the database servers in the database 320 run a DBMS 360, 460, 370, 380, and each of the DBMSs preferably include an optimiser 365, 465, 375 and 385. The various servers need not all have the same type of DBMS. For instance some database servers may have an IBM DB2 DBMS, whilst others may have an Oracle 7 DBMS. In the example, database servers 330, 340 and 350 are DB2 servers, whilst database server 450 is an Oracle 7 server.

In the example shown in Figure 4A the database is a distributed database, each database server having its own storage devices 367, 467, 377 and 387 respectively. However the invention is equally applicable to a single parallel database arrangement in which storage devices are

It is the server 400 to which the client computer 300 is connected that forms the system of the preferred embodiment. This server also includes a DBMS 410 having an optimiser 420. This DBMS is used to manage a storage device 430 having a data dictionary stored therein. The data dictionary contains "metadata" indicating how the data in the database 320 is distributed among the various servers 330, 450, 340 and 350.

In addition to the DBMS, the server 400 includes a device 440 for determining how to decompose any queries received by the server 400 from the client computer 300 (and indeed any other computers which may be connected to the server 400). This device 440 will be referred to hereafter as the "Parallel Query Decomposer" or "PQD", and is preferably embodied as software routines. The PQD 440 refers to the data dictionary on storage device 430 in order to determine how to decompose incoming queries from client computers.

Before discussing the preferred embodiment in more detail it should be noted that although the server 400 is shown in Figure 4A as a physically separate entity, it is not necessary for the server 400 to be physically separate. It may, for example, be incorporated in one of the database servers 330, 450, 340 or 350, or indeed may reside on the client computer 300. The important feature is that the database server 400 is logically positioned between the client computer 300 and the database 320.

The system of the preferred embodiment operates in the following way. SQL queries sent from one of a number of possible sources, for example an application 310 on client computer 300, are received by the PQD device 440 of server 400. PQD then transforms each query into a number of parallel SQL queries based on its own interrogation of the data dictionary 430, or alternatively can use a local optimiser on one of the database servers to do some of the transformation where there are several database servers of the same type; this latter approach is discussed later with reference to Figure 4B.

Once the PQD has transformed the incoming SQL query, it then migrates the resulting SQL subtasks to the relevant database servers of the database 320 for subsequent execution. PQD can perform the migration of these subtasks either explicitly using its server's own transport database servers to despatch remote tasks across the DBMS's own distribution medium. It then recombines the results, or allows the local optimiser to do so if possible, depending on the distribution strategy chosen.

In Figure 4A, PQD performs all of the decomposition of the original SQL query Q based on the metadata in the storage device 430. PQD 440 accesses the metadata by extracting the table names from the query Q and then querying the DBMS 410 using those table names. In effect PQD asks the DBMS 410 whether it has any information stored about these tables. The DBMS 410 then returns the metadata to PQD 440 as the result of that query. PQD uses the metadata to decompose the original query Q, and then passes the constituent queries to a communications layer 445. The communications layer uses the standard transport mechanism (eg TCP/IP) of the server 400 to distribute the resulting subtasks (which are still in SQL format) to the database servers 330, 450, 340 and 350 of the database 320. Since the subtasks are still in a standard query language (in this case SQL) the optimisers on each of the database servers can understand the queries sent to them by the server 400, even though they may not each be of the same type. The optimisers perform the subtasks directed to them with reference to the data stored in their server's respective storage devices, and then return the results to the server 400. Here PQD recombines them and returns them to the client application 310.

In Figure 4B, the arrangement is exactly the same, but in this case the PQD 440 is adapted to take advantage of the facilities offered by the various optimisers on the database servers. On receiving metadata from the data dictionary in storage device 430, PQD notes that three of the database servers 330, 340, 350 are DB2 servers, and hence that an optimiser on one of those servers will be able to distribute the query across those servers based on that optimiser's interrogation of its own data dictionary.

Hence in the Figure 4B case, PQD decomposes the original SQL query into two sub-tasks, one to be sent to the optimiser of database server 330, and one to be sent to the non-DB2 database server 450 (in the example this is an Oracle 7 server). On receipt of the subtask, optimiser 365 in database server 330 refers to its own data dictionary in storage device 367 and transforms the sub-task into further sub-tasks to be performed by itself and/or by one or more of the other DB2 servers in the application 310 is found to exist on all of the DB2 servers 330, 340, 350, and so the optimiser 365 creates three sub-tasks, one of which it performs locally and the other two of which are sent to the other database servers 340 and 350. The database servers 340 and 350 execute these sub-tasks in the conventional manner and return their results to the database server 330. The optimiser 365 recombines these results with its own results and sends the result back to the server 400. Meanwhile the database server 450 has executed its sub-task and has also returned the results to the server 400. PQD 440 then recombines these two result sets and returns them to the application 310, thereby completing the execution of the original query Q.

A significant strength of the technique described above with reference to Figures 4A and 4B is that, unlike the prior art approaches, it is capable of running across heterogeneous servers, which may be running on different hardware but also running different DBMSs, e.g. DB2/6000, Oracle, etc. A further strength of this technique is that it provides the ability to support different parallel and distributed architectures, and also allows flexibility in the choice of whether to allow the DBMS of a database server to migrate subtasks, or whether instead to use PQD and its server's own independent transport mechanism, in which case PQD will be fully responsible for recombination.

In summary the system of the preferred embodiment parses SQL queries and automatically transforms them into modified queries which take advantage of data parallelism inherent in the structure of the data being queried. The modified queries are also represented in SQL.

From the above it is clear that the system of the preferred embodiment is able to interact with heterogeneous databases in a manner which is completely transparent to the client application. However the same technique can still be used when the database is homogeneous, and in such cases the procedure is simplified still further. The homogeneous case will now be described with reference to Figure 4C.

In Figure 4C, the arrangement of the server 400 is physically the same as that illustrated in Figures 4A and 4B. In this case, however, the database 320 consists solely of Oracle 7 database servers, namely database servers 330, 340 and 350. Further the DBMS on server 400 is an Oracle 7 DBMS, this not necessarily being the case in figures 4A and 4B.

410 in the normal manner to see whether there is any information stored about the table names contained within the query. In this case the DBMS 410 notes that it has aliases (or synonyms as they are referred to in Oracle 7) stored in the data dictionary for the table names in question. This is all that is needed in the homogeneous case to identify the location of the various partitions of a particular table across the database servers within the database. Hence, instead of the full metadata being returned to PQD, the DBMS merely sends the synonyms back to PQD.

PQD 440 uses these synonyms to decompose the query into a number of constituent queries and then returns the constituent queries to the optimiser 420 of the Oracle DBMS 410. The optimiser passes the constituent queries to a communications layer 415 within the DBMS 410, which then uses the transport mechanism of the DBMS (SQL*Net in the case of an Oracle 7 DBMS) to dispatch the queries to the relevant database servers 330, 340, 350 within the database 320. These queries are processed by the optimisers 365, 375, 385 within the various database servers 330, 340, 350, and the results are returned to the server 400. Here the optimiser 420 assimilates the data into a single response for communication back to the client and sends that response to PQD 440, PQD 440 performing the act of dispatching the response back to the client application 310.

From the above description of Figures 4A-4C, it can be seen that PQD accepts queries and checks whether the tables referred to in those queries have been partitioned across multiple servers. Preferably it does this by interrogating the data dictionary in storage device 430 and looking for synonyms to partitioned tables which bear the same name as the table in the original query (if the database is homogeneous), or looking for metadata (if the database is heterogeneous). Alternatively explicit partitioning information from a table in a publicly accessible tablespace can be used. PQD automatically transforms the original query into a compound query which contains multiple parallel queries, with one for each of the partitions identified from the data dictionary (unless the optimisers in the database servers are utilised to perform some of the decomposition). This takes the responsibility away from the application 310, which can treat a table as a single entity with no regard for whether the table may have been partitioned across a number of parallel database servers. Furthermore, the DBA retains the freedom to migrate data from one database server to another, and to change the

the data dictionary 430 to reflect the partitioning of the data, either through creation of synonyms, or explicit partitioning information, the DBA enables PQD to transparently and automatically track the location of all parts of the original table. Since the data definition is performed using SQL statements, and any synonyms are created using SQL statements it is straightforward to automate the above process via the creation of administration tools. These tools would then automatically update the synonyms when the tables are partitioned.

As already mentioned, PQD can operate by accessing the Data Dictionary 430 on the PQD server 400, and looking for synonyms or metadata which relate to partitions of the target table(s). When partitioning a table, the DBA would therefore create the necessary synonym or metadata for each partition. In the case of synonyms, each synonym indicates the database server on which a partition resides by referring to a database link for that database server.

The technique of the preferred embodiment will now be described further with reference to an example in which PQD is used to process queries to a homogeneous database, in this case the environment being the Oracle DBMS.

The data and query used in this example are trivially small, and it is not suggested that the partitioning of such a table would be a good idea. The example is in two parts. First the query is issued by the PQD server 400 to a single database server, and secondly by the PQD server to a pair of database servers across which a table has been partitioned.

In both parts of the example, the client application 310 is running on a workstation 300 different from the workstations that are acting as servers. However this is not essential and the invention can also operate in situations where the client application is running on one of the database servers.

In this example synonyms are used as the mechanism for recording how the data has been partitioned. This is only illustrative of one mode of operation of PQD, and other methods can be used where appropriate (e.g. a metadata table in system tablespace containing the partitioning information of base tables).

First we will consider the situation in which a query is issued to the PQD server 400 by the client application 310 and is subsequently processed by a single database server, the query being as follows:

Query: select * from people;

The data stored on the database server is shown in Table 1 below, whilst the synonym table stored in the data dictionary of the PQD server 400 is shown in Table 2 below:

TABLE 1 : PEOPLE

NAME	AGE	HEIGHT
Adam	37	178
Benjamin	64	168
Charles	48	183
David	53	175
Edward	28	178
Frederick	41	197

TABLE 2 : ALL_SYNONYMS

SYNONYM_NAME	TABLE_NAME	DB_LINK
people	people	server
another_synonym	another_table	another_dblink

The query is despatched by the Oracle DBMS on PQD server 400 to the database server, which processes it and returns the result to the PQD server. PQD then passes the result back to the client application 310.

If, however, the people table is to be split across two database servers which are known by the db_links server1 and server2, then the situation is more complicated. The DBA may decide to split the table horizontally into two alphabetic groups, for example by creating a PEOPLE table on each of the two database servers, each table then being populated with the records with surnames in the half of the alphabet assigned to that table. Tables 3 and 4 below shows the contents of the PEOPLE tables on the two database servers as a result of the split by the DBA:

TABLE 3 : PEOPLE on server 1

NAME	AGE	HEIGHT
Adam	37	178
Benjamin	64	168
Charles	48	183

TABLE 4 : PEOPLE on server 2

NAME	AGE	HEIGHT
David	53	175
Edward	28	178
Frederick	41	197

The DBA issues the following commands at the PQD server 400, so that the data dictionary at the PQD server 400 contains the synonyms for the table partitions:

```
drop database link server1;
create database link server1 using '<connect_string_1>';
drop database link server2;
create database link server2 using '<connect_string_2>';
```

(where <connect_string_1> and <connect_string_2> are replaced by valid connect strings for each of the two database servers, for use by Oracle's SQL*Net product)

```
drop synonym people1;
create synonym people1 for people@server1;
```

```
drop synonym people2;
create synonym people2 for people@server2;
```

As a result of these commands the synonym table in the data dictionary of PQD server 400 will have the following entries:

TABLE 5 : ALL_SYNONYMS at PQD server		
SYNONYM_NAME	TABLE_NAME	DB_LINK
people1	people	server1
people2	people	server2
another_synonym	another_table	another_dblink

The client application 310 will still issue the same query Q as it did before the table was split, ie:

Query: select * from people;

but because there are now synonyms for the PEOPLE table, PQD automatically decomposes the query into the following query:

PQD Query: select * from people1 UNION select * from people2;

The process used by the system of the preferred embodiment in this current example will now be discussed with reference to Figure 5A, which is a flow diagram generally illustrating the process carried out in the preferred embodiment of the present invention if the database is homogeneous. The query Q issued from the client computer 300 is first received by PQD 440 at step 510. Then, at step 520, PQD consults the metadata (in this case the synonym table) in the data dictionary of the storage device 430, and based on that interrogation transforms the query into a compound query at step 530. Using the above example, the compound query is then despatched by the optimiser 420 (via communications layer 415) on PQD server 400 to one of the database servers 1 and 2, and the server 400 then awaits the results from the database servers (step 550). Since we are currently considering the situation in which the database is homogeneous, the compound query need only be sent to one of the two database servers, since the optimiser on the recipient database server will be able to pass on subtasks to the other database server. The optimiser on that recipient database server will also recombine the results from the other database server with its own results before dispatching the result to the PQD server 400. Hence the PQD server 400 can simply forward the results (step 560) directly to the client application 310 once it receives them from the database server.

It is important to note that the application 310 remained unchanged in this last example, despite the fact that the DBA added a second database server to the database and horizontally partitioned the PEOPLE table across the two database servers.

The above example has illustrated the use of PQD in a homogeneous environment, and demonstrates how PQD can make use of the native optimiser of one of the database servers in such circumstances. By modifying the query appropriately, PQD can influence the execution plan that the optimiser produces, and hence control the parallelism employed in processing the query. It also makes use of the DBMS on one of the database servers to produce the final result set.

In other configurations however, PQD can use explicit partitioning information in the data dictionary 430 to decompose a complex query and execute the resultant queries across either a homogeneous or heterogeneous parallel database. In the heterogeneous case, it is not possible to make use of the DBMS on one of the database servers to produce the final result set, and a mechanism has to be provided by PQD for combining the intermediate result sets from sub-queries to dissimilar DBMSs. In such cases, PQD need only generate as many sub-queries as there are dissimilar DBMSs, treating any number of similar DBMSs as a homogeneous sub-server; this principle was illustrated earlier with reference to Figure 4B.

The process carried out at the PQD server 400 in the case of a heterogeneous database is discussed further with reference to Figure 5B. As before the query from the client application 310 is received by PQD 440 of server 400 (step 600), and the metadata in data dictionary 430 is consulted (step 610). Preferably the metadata takes the form of a table containing the fields 'Table_name', 'Partition_name', and 'Server_name'. The table name will be the name as used in the original query Q, whilst the partition name will be the name of the partition stored on a enable a communications link to be established with the database server containing the partition of the original table.

As a result of consulting the metadata, PQD transforms the original query Q into a number of sub-queries at step 620. As mentioned above, in preferred embodiments one sub-query is generated for each type of server in the database; hence in the Figure 4B example two sub-queries are generated rather than four. The subqueries are despatched to the relevant servers at step 630, and once the results are returned (any assimilation that can have been performed by the DBMSs of the database servers having taken place), PQD 440 assimilates the result sets at step 640. If necessary PQD then performs at step 650 any aggregation that may be required by the standard SQL aggregations, including, but not limited to, SUM(), AVG() or STD(). Once this has been done the results are then returned to the client application at step 660.

The technique used by PQD to perform aggregation operations within queries which are decomposed will now be discussed in a little more detail. All the standard SQL aggregations (including, but not limited to, SUM(), AVG() and STD()) can be performed in a decomposed query by adding additional fields to the intermediate result sets. For example, the AVG(age) of all people in a partitioned table may be derived simply by forming the following result set for each partition:

```
insert into IntermediateResult (pcount,psum)
  select count(age), sum(age) from People<i>;
```

where People<i> is (in the homogeneous case, as in the example above) the synonym of the partition being interrogated, and then forming the overall result by computing:

```
select sum(psum)/sum(pcount) from IntermediateResult;
```

It will be apparent to those skilled in the art that similar techniques can be used to generate any of the standard aggregations although different intermediate fields are required for some, such as "sum of squares" for computing STD() or VAR().

It can be seen from the above description that the system of the preferred embodiment of the present invention uses the power of parallel transparency to the client, heterogeneous operation, isolation between applications and data, and a choice of parallel architectures. The combination of these attributes make the system much more flexible than any of the prior art techniques.

CLAIMS

1. A system for processing requests from a client application (310) on a client computer (300) for retrieval of data from a database (320) having a plurality of database servers (330, 450, 340, 350) on which data is stored, the system comprising:

a storage means (430) for storing information identifying how the data is distributed across the plurality of database servers;

a first processing means (440), interposed between the client computer (300) and the database (320), for receiving a request from the client computer in a standard query language, and with reference to the storage means (430), transforming the request into one or more constituent requests in the standard query language;

communication means (445; 415) for transmitting the constituent requests to the associated database servers;

a second processing means (440; 410, 420) being arranged to receive the data sent from the associated database servers as a result of the constituent requests, and for assimilating that data as a single response for communication back to the client computer; and

dispatch means (440) for sending the single response to the client application.

2. A system as claimed in Claim 1, wherein the first processing means produces one constituent request for each type of database server in the database.

3. A system as claimed in Claim 2, wherein the associated database server further divides the constituent request so as to send one request to each database server of the same type as the associated database server.

4. A system as claimed in Claim 3, wherein the associated database server further recombines results from the database servers of the same type before sending the data to the second processing means.

5. A system as claimed in any preceding claim, wherein if all the database servers in the database are of the same type, then the first processing means uses aliases stored in the storage device to create the constituent requests.

6. A system as claimed in any of claims 1 to 4, wherein if some of the database servers in the database are of different types, then the first processing means uses metadata stored in the storage device to create the constituent requests.

7. A system as claimed in any preceding claim, wherein the second processing means further comprises aggregation means for performing any necessary aggregation of the data received by the second processing means from the database servers.

8. A method of processing requests from a client application (310) on a client computer (300) for retrieval of data from a database (320) having a plurality of database servers (330, 450, 340, 350) on which data is stored, the method comprising the steps of:

storing information in a storage means (430) for identifying how the data is distributed across the plurality of database servers;

employing a first processing means (440), interposed between the client computer (300) and the database (320), to receive a request from the client computer in a standard query language, and with reference to the storage means (430), transforming (620; 530) the request into one or more constituent requests in the standard query language;

transmitting (630; 540) the constituent requests to the associated database servers;

receiving the data sent from the associated database servers as a result of the constituent requests, and assimilating (640) that data as a single response for communication back to the client computer; and

sending (660; 560) the single response to the client application.

9. A method as claimed in Claim 8, wherein the first processing means produces one constituent request for each type of database server in the

10. A method as claimed in Claim 9, wherein the associated database server further divides the constituent request so as to send one request to each database server of the same type as the associated database server.

11. A method as claimed in any of claims 8 to 10, further comprising the step of, subsequent to the assimilation step (640), performing any necessary aggregation (650) of the data received from the database servers.

22.

Patents Act 1977
Examiner's report to the Comptroller under Section 17
(The Search report)

Application number
GB 9500252.3

Relevant Technical Fields

(i) UK Cl (Ed.N) G4A (AUDB)

(ii) Int Cl (Ed.6) G06F

Search Examiner
M J [illegible] IS

Date of completion of Search
9 FEBRUARY 1995

Databases (see below)
(i) UK Patent Office collections of GB, EP, WO and US patent specifications.

(ii)

Documents considered relevant following a search in respect of Claims :-
1-11

Categories of documents

X: Document indicating lack of novelty or of inventive step.

Y: Document indicating lack of inventive step if combined with one or more other documents of the same category.

A: Document indicating technological background and/or state of the art.

P: Document published on or after the declared priority date but before the filing date of the present application.

E: Patent document published on or after, but with priority date earlier than, the filing date of the present application.

&: Member of the same patent family; corresponding document.

Category	Identity of document and relevant passages	Relevant to claim(s)
X	EP 0625756 A1 (HUGHES AIRCRAFT) whole document	1, 8 at least

Databases: The UK Patent Office database comprises classified collections of GB, EP, WO and US patent specifications as outlined periodically in the Official Journal